

Atlas LSH Neural Networks as Compositional Decision Trees

Executive Summary

Atlas LSH neural networks can be viewed as **compositional decision trees with extreme parameter reuse**.

A locality sensitive hash (LSH) computed from the input generates a binary code

$$b(x) \in \{0,1\}^k$$

which determines the computational route taken through the network.

In the simplest toy architecture, each bit selects one matrix from a pair:

$$W_i^{(0)}, W_i^{(1)}.$$

The resulting computation is

$$F_{b(x)} = W_k^{b(k)} W_{k-1}^{b(k-1)} \dots W_1^{b(1)} x.$$

Although only $2k$ matrices are stored, the network implicitly defines

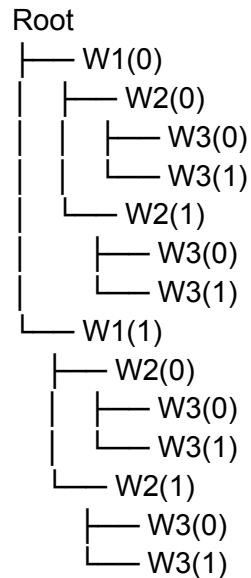
$$2^k$$

possible computational pathways.

This viewpoint suggests that Atlas systems may be better understood as **hierarchical trees of reusable operators** rather than conventional feedforward networks.

1. Tree View

The simplest Atlas architecture corresponds to a depth-(k) binary tree.



Unlike classical decision trees, operators are heavily reused:

- each operator participates in many complete routes,
- each training example updates only the selected route,
- specialization emerges through repeated route-specific updates.

Larger Atlas systems may consume multiple hash bits per level:

m bits $\rightarrow 2^m$ branches.

This yields decision trees with enormous branching factors.

2. Hypothesis: Specialization by Depth

A natural question:

Do different levels of the tree learn qualitatively different roles?

This has not yet been experimentally established.

However, optimization dynamics suggest the following hypothesis.

Depth	Possible role
Early	coarse routing, context setting
Middle	representation construction
Deep	fine specialization, corrections
Final	output calibration

Because upper-level operators participate in many routes, they receive diverse gradients and may become generalists.

Deep operators participate in fewer routes and may become specialists.

Possible examples:

- vision: scene → object → fine category,
- language: style → topic → concept,
- multimodal: modality → semantic domain → detail.

3. Questions for Researchers

Before reading further, pause and ask:

Q1. In your own sparse MoE or conditional computation systems, have you already observed depth-dependent specialization?

Q2. Are experts near the root more reusable than experts near the leaves?

Q3. Does expert reuse frequency follow a power law?

Q4. Do early experts become "general infrastructure" while late experts become niche specialists?

Many existing training logs may already contain the answers.

4. Mutual Information Hypothesis

A second hypothesis is that operator semantics vary systematically with depth.

For operator O_d at depth d , define

$$I(O_d ; Y)$$

as the mutual information between operator selection and some task variable Y .

Examples of Y :

- class label,
- topic,
- modality,
- task identity,
- difficulty,
- error type.

Hypothesis:

$$I(O_d ; Y)$$

changes with depth.

Possible outcomes:

Hierarchical specialization

Early operators:

$$I(O_d ; \text{modality})$$

Late operators:

$$I(O_d ; \text{fine class})$$

No hierarchy

Mutual information remains approximately constant across depth.

Emergent hierarchy

Different semantic variables dominate at different depths.

5. Suggested Experiments

Experiment 1: Route Statistics

Measure:

- route frequencies,
- operator usage frequencies,
- route overlap distributions.

Question:

Are there universal scaling laws?

Experiment 2: Depth versus Reuse

For each operator measure:

- activation frequency,
- gradient magnitude,
- update count.

Question:

Does specialization increase with depth?

Experiment 3: Mutual Information Profiling

For each depth d :

$I(O_d ; Y)$

for multiple task variables.

Question:

Does a semantic hierarchy emerge automatically?

Experiment 4: Representation Similarity

Measure CKA or representational similarity across operators.

Question:

Are shallow operators more generic than deep operators?

6. A Challenge

Many researchers already possess trained sparse models.

The experiments proposed here require little or no retraining.

Before building a new Atlas architecture, ask:

What hidden tree already exists inside your current model?

The answer may already be stored in your checkpoints.